

Counting small cuts in a graph

Barbara Geissmann and Rastislav Šrámek

Institute of Theoretical of Computer Science, ETH Zurich, Zurich, Switzerland
e-mail: geibarba@student.ethz.ch, rsramek@inf.ethz.ch

Abstract. We study the minimum cut problem in the presence of uncertainty and show how to apply a novel robust optimization approach, which aims to exploit the similarity in subsequent graph measurements or similar graph instances, without posing any assumptions on the way they have been obtained. With experiments we show that the approach works well when compared to other approaches that are also oblivious towards the relationship between the input datasets.

1 Introduction

Dealing with uncertainty is an ever more common problem. We are flooded with data recorded by virtually all modern devices from cars to cellular phones, data about various networks, observations of different phenomena. In order to be able to extract meaningful information from this data, we need to be able to remove or at least identify the noise that is inherently present, whether due to measurement errors or due to systematic influence of unknown factors. In this paper we consider a novel method of robust optimization introduced by Buhmann et al. [1], and apply it to the problem of searching for the global minimum cut in a graph.

Finding the global minimum cut in a graph is a well studied problem with applications ranging from information retrieval [2] to computer vision [3]. The problem is to separate the set of graph vertices V into two non-empty disjoint sets X and $X \setminus V$, such that the sum of the weights of edges that have one end-point in X and another in $X \setminus V$ is minimized. Since a cut is fully determined by the subset X , we will denote it only by X with the possible caveat that X and $V \setminus X$ denotes the same cut. We are in particular interested in the minimum cut as a measure of network robustness [4]: If the weight of an edge represents the effort needed to cut that particular edge, the minimum cut represents the least effort necessary to disconnect the graph.

Suppose that we are looking for a minimum cut in a graph, for instance one that represents connections between nodes in a sensor network. However, instead of the “true” graph we are only given two snapshots of it from two different points in time, with the same topology, but with different edge-weights. What should we do in order to identify a minimum cut in the “true” underlying graph, or a cut that will be minimum in a third, similar, snapshot?

It is clear that without very precise understanding of the process by which we obtain the graph measurements, we are unable to answer this question with full confidence and thus any solution will be only an heuristic. Nevertheless, the setting is a realistic and very common one, and we should not give up. For instance, one intuitive course of action would be to average the weights provided by two instances edge by edge and compute a minimum cut on the resulting graph. In this paper we want to show that a different method, also oblivious to the properties of the data generator, might yield better results.

1.1 Approximation set optimization

We will introduce the aforementioned robust optimization method of Buhmann et al. [1] in greater detail. We will refer to it as approximation set optimization. Recall that the weight of a graph cut X is the sum of the weights of the edges that have one endpoint in X and another in $V \setminus X$, we will denote it by $w(X)$.

Definition 1 (ρ -Approximate Cut).

Let $\lambda(G)$ denote the weight of a global minimum cut in G . For a parameter $\rho \geq 1$, a ρ -approximate cut X is a cut with weight at most $\rho\lambda(G)$, $w(X) \leq \rho\lambda(G)$.

Definition 2 (ρ -Approximation Set).

A ρ -approximation set of G , denoted by $A_\rho(G)$, is the set of all ρ -approximate cuts in G , $A_\rho(G) = \{X \in V | w(X) \leq \rho\lambda(G)\}$.

Let G_1 and G_2 be two weighted graphs with the same topology but different edge-weights. The approximation set optimization method states that we should find a factor ρ , for which the intersection of the ρ -approximation sets $A_\rho(G_1) \cap A_\rho(G_2)$ is the largest, when compared to the expected size of this intersection if the instances were generated at random. We then pick a solution at random from the intersection of the resulting ρ -approximation sets. Formally, we look for ρ^* such that

$$\rho^* = \arg \max_{\rho} \frac{|A_\rho(G_1) \cap A_\rho(G_2)|}{Es(|A_\rho(G_1)|, |A_\rho(G_2)|)}, \quad (1)$$

where $Es(|A_\rho(G_1)|, |A_\rho(G_2)|)$ is the expected size of the intersection of the ρ -approximation sets of the given size. We call the value

$$|A_{\rho^*}(G_1) \cap A_{\rho^*}(G_2)| / Es(|A_{\rho^*}(G_1)|, |A_{\rho^*}(G_2)|) \quad (2)$$

unexpected similarity. It is a measure of similarity of G_1 and G_2 , with respect to the optimization problem of looking for the minimum cut.

In order to successfully apply the method, we need to be able to solve five problems: Count the number of ρ -approximate cuts in a graph G , count the number of cuts in the intersection of the approximate sets of two graphs G_1 and G_2 , compute

the function for the expected intersection Es , find the optimal factor ρ^* , and choose a cut at random from the set of all cuts that are ρ -approximate for the graphs G_1 and G_2 at the same time.

1.2 Related work

Robust optimization is a widely studied subject. However, in order to be able to derive provably optimal methods, one needs to restrict the scope of inputs, or make other strong assumptions about them. For instance *Stochastic optimization* [5, 6] and *Robust optimization* [7] expect that we know respectively the complete distribution of an instance and the complete set of instances. Various methods of optimization for stable inputs on the other hand suppose that the input cannot change too much [8–10]. In our case, by not assuming anything about the input we lose the ability to apply any of these but greatly increase the scope of problems for which we can hope to achieve good solutions.

The remainder of the paper is structured as follows. In Section 2 we show how to count the sizes of ρ -approximation sets of cuts and their intersections, in Section 3 we will derive a approximate formula for the expected size of the approximation set intersection on random instances, followed by experimental evaluation of the method in Section 4 and concluding remarks in Section 5.

2 Algorithms for counting small cuts

For many combinatorial optimization problems, the problem of counting approximate solutions is #P-complete, even if the optimization problem itself is efficiently solvable. The reason for this lies in the possibly exponential number of solutions. For instance, there can be n^{n-2} short spanning trees in a graph with n vertices or 2^{n-2} short s - t paths in a directed acyclic graph [11]. For minimum cuts, however, the possible number of near-optimal cuts is small. Dinits et al. [12] showed that there can be at most $\binom{n}{2} = O(n^2)$ minimum cuts in a graph and Karger [13] showed that the number of ρ -approximate cuts is at most $O(n^{2\rho})$. This makes our life significantly easier, since we can afford to enumerate, not only count the cuts in the approximation sets. Note that calculating the number of cuts shorter than an arbitrary threshold is still #P-complete [14]. This is not surprising, since with rising threshold the problem must turn from easy to difficult, as calculating the maximum graph cut is a NP-complete problem.

There are at least two different algorithms that can compute the ρ -approximation sets of a graph. One is by Nagamochi, Nishimura and Ibaraki [15] and it solves the task deterministically in $O(mn^{2\rho})$ time if m is the number of edges of the graph with n vertices. The other is an adaption of the *recursive contraction algorithm* by Karger and Stein [16], and it finds all ρ -approximate cuts in $O(n^{2\rho} \log^3 n)$ time with high probability.

We will use the approach of Karger and Stein because it is the fastest currently known algorithm. Apart from that it allows us to make an adaptation with which we can directly compute the approximation sets.

2.1 Karger and Stein's Algorithm

The recursive contraction algorithm, described in Algorithm 1, finds a minimum cut in a graph as follows. The main idea is to repeatedly choose an edge at random and contract it, which means that the two end vertices of this edge are merged into a single vertex. The algorithm starts with two copies of the graph. On each of them it performs random edge contractions until the graph has shrunk down to a certain size. Then the graph is copied again and the algorithm continues, again on both graphs. When only two vertices remain, the edges contracted into each of the two vertices correspond to one set of vertices in a cut and the weight of the remaining edge corresponds to the cost of this cut. The algorithm keeps track of the found cuts and the best cut is returned. The intuition behind the algorithm is that in the beginning, the probability of contracting an edge from a minimum cut, and thus excluding this cut from the set of possible results, is low. As the algorithm progresses, this chance increases, but this is combated by the increased number of concurrent evaluations.

Algorithm 1 (Recursive Contraction Algorithm).

```

RECURSIVECONTRACT( $G$ )
if  $|V| \leq 6$  then
     $G \leftarrow \text{CONTRACT}(G, 2)$ 
    return the cut
else
    repeat twice
         $G' \leftarrow \text{CONTRACT}(G, \lceil n/\sqrt{2} + 1 \rceil)$ 
        RECURSIVECONTRACT( $G'$ )
    return the smaller cut
end

```

The routine $\text{CONTRACT}(G, x)$ does repeated edge contraction in G until only x vertices remain. The whole algorithm runs in $O(n^2 \log n)$ time. The probability that it finds a particular minimum cut is at least $\Omega(1/\log n)$. If we repeat the algorithm $O(\log^2 n)$ times, we will find any particular minimum cut with high probability. The algorithm can be adjusted so that it returns all minimum cuts that it finds instead of only one. Since the total number of unique minimum cuts in a graph is bounded from above by $\binom{n}{2}$, we can find every minimum cut with high probability, within the total time complexity of $O(n^2 \log^3 n)$.

Karger and Stein's algorithm can be modified to find all ρ -approximate cuts [16], by changing the reduction factor from $\lceil n/\sqrt{2} + 1 \rceil$ to $\lceil n/\sqrt[2\rho]{2} + 1 \rceil$ and stopping the contraction when 2ρ vertices remain. In this case, all remaining possible cuts are evaluated. The running time increases to $O(n^{2\rho} \log n)$, whereas the success probability remains the same. Since the number of ρ -approximate cuts is bounded by $\Theta(n^{2\rho})$, we can find all ρ -approximate cuts with high probability by repeating the algorithm $O(\log^2 n)$ times. This gives the overall time of $O(n^{2\rho} \log^3 n)$.

2.2 Approximation Set Optimization Algorithm

Recall that we want to determine ρ that maximizes the unexpected similarity of the two graphs G_1 and G_2 with respect to the minimum cut problem. To this end we first need to compute the ρ -approximation sets of G_1 and G_2 and their intersection. The former is done by `APPROXIMATIONSET(G)`, which is an adapted version of Karger and Stein's recursive contraction algorithm. Afterwards, we are ready to compute the expected and unexpected similarity, Es and u_sim . By sampling for the best ρ we find the intersection of $A_{\rho^*}(G_1)$ and $A_{\rho^*}(G_2)$ from which we can pick a cut at random, as a solution that generalizes for both instances. The whole process is described in Algorithm 2.

Algorithm 2 (Approximation Set Optimization Algorithm).

```

for  $\rho = 1$  until  $\rho = \text{MAX}$  do
   $A_{\rho}(G_1) \leftarrow \text{APPROXIMATIONSET}(G_1, \rho)$ 
   $A_{\rho}(G_2) \leftarrow \text{APPROXIMATIONSET}(G_2, \rho)$ 
   $\text{intersection} \leftarrow \text{intersect}(A_{\rho}(G_1), A_{\rho}(G_2))$ 
   $u\_sim \leftarrow |\text{intersection}| / Es(|A_{\rho}(G_1)|, |A_{\rho}(G_2)|)$ 
  if  $u\_sim > \text{max\_sim}$  then
     $\text{max\_sim} \leftarrow u\_sim$ 
     $\rho^*\_intersection \leftarrow \text{intersection}$ 
  end
end

```

Now let us discuss some issues of the algorithm in more detail and derive its time complexity. Karger and Stein's version of the algorithm returns the cuts in an implicit way. Since we want to be able to compute the intersection of the approximation sets of two different graphs as well as to choose a cut from the intersection and apply it to a third graph, we need them explicitly. One simple possibility to meet this requirement is to store for each vertex whether it is in the cut or not. The entire cut can then be represented as a bit string of length n . Notice, that this notation is ambiguous, since the inverse of a bit string describes the same cut. We can fix this by allowing only cuts that have the first bit set to zero.

By treating the bit strings as numbers, we can sort the cuts in the approximation sets and then build the intersection in a merging fashion in $O(n^2 \log n)$ time, since the number of cuts in each approximation set is bounded from above by $\Theta(n^{2\rho})$.

Returning the cuts in an explicit manner also implies extra work during the computation of the approximation sets. After every recursion phase the union of the two found approximation sets is returned. To overcome difficulties like different smallest cut weights and duplicates, one possibility is to again sort the cuts. This extra work requires a factor of $O(\log n)$ additional time for the entire approximation set algorithm. So we end up with an approximation set algorithm that takes $O(n^{2\rho} \log^4 n)$ time.

The time to compute u_sim , the unexpected similarity, depends on the complexity of the function Es . We postpone this to Section 3.

The last thing we have to look at is the range and step size of the values for ρ in the for loop. To choose a good bound for the largest ρ we want to test is not easy. It depends a lot on the structure of the graphs and the range of their weights. Therefore, we may want to start with rather big steps and refine them as we go.

3 Expected intersection size

Having described an algorithm that counts the size of individual approximation sets and their intersection, we turn to the question of deriving a formula for the expected size of the intersection of the approximation sets. For a more detailed exposure we refer the reader to Chapter 3.2 of the bachelor thesis of Barbara Geissmann [17].

For the expected similarity, we will only consider cuts on complete graphs. Otherwise we would need to track whether each cut X cuts the graph into only two parts, since the Karger-Stein algorithm and its modifications return only such cuts.

We first show that an arbitrary subset of cuts does not necessarily have to form a valid approximation set.

Definition 3 (Crossing Cuts). *Two cuts X and Y cross each other if $X \cap Y \neq \emptyset$, $X - Y \neq \emptyset$, $Y - X \neq \emptyset$, and $V - X - Y \neq \emptyset$.*

Definition 4 (Composed Cuts). *Let X and Y be two cuts that cross each other. Then they must define four further cuts:*

$$\begin{aligned} Z_1 &= X \cap Y & Z_2 &= X - Y \\ Z_3 &= Y - X & Z_4 &= V - X - Y \end{aligned} \tag{3}$$

We call Z_1 , Z_2 , Z_3 , and Z_4 the composed cuts of X and Y .

Theorem 1. *If two cuts X and Y in the approximation set $A_\rho(G)$ cross each other, then at least two of the four composed cuts of X and Y have to be in $A_\rho(G)$ as well.*

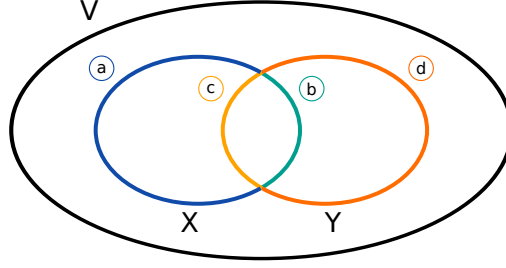


Fig. 1: Crossing cuts

Proof. According to the Figure 1 we denote by a , b , c , and d the sums of the weights of the cut edges between the sets Z_1 , Z_2 , Z_3 , and Z_4 , as in Definition 4. Without loss of generality, let us suppose that $a \leq b$, $c \leq d$, $b \leq d$. Then, for cuts X and Y to be in the approximation set, there must be a threshold $t := \rho\lambda(G)$ such that $a + b \leq t$ and $c + d \leq t$. The 4 composed cuts will have weights $a + c$, $a + d$, $b + c$, and $b + d$. However, it must hold that $a + c \leq t$ because both a and c are at most $t/2$, and also $b + c \leq t$ because we can replace d in $c + d \leq t$ with b which is at most as large. \square

Theorem 1 shows that not every subset of cuts forms a feasible approximation set. Using Theorems 1 and 2 from [1] we can conclude that the expression $|A_\rho(G_1)||A_\rho(G_2)|/|S|$, where S denotes the set of all cuts, is a lower bound on the expected size of the intersection, but not its true value.

We see that as soon as we have crossing cuts, we lose freedom in the number of cuts which we can freely choose. We will show that this loss of freedom is substantial enough that by restricting ourselves to approximation sets without crossing cuts we get a approximation of the true expected value.

The number of ways in which we can cut a graph of n vertices m times so that the cuts do not cross is equal to the number of ways we can partition a set of n integers into $m + 1$ non-empty subsets. The latter describes the well known Stirling number of the second kind, denoted by $\left\{ \begin{smallmatrix} n \\ m+1 \end{smallmatrix} \right\}$, and defined by the explicit formula

$$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n. \quad (4)$$

Observe that in a complete graph, a non-empty cut on n vertices can be chosen in $2^{n-1} - 1$ ways. Furthermore, the smallest approximation set that can contain crossing cuts is of size 4. Such an approximation set would be approached by $\left\{ \begin{smallmatrix} n \\ 5 \end{smallmatrix} \right\}$. We conclude that at least if the number of vertices n is large compared to the number of cuts in the approximation set, the loss of freedom to choose 2 additional cuts

significantly outweighs the additional flexibility we gained by choosing the second cut in $2^{n-1} - 2$ ways. Note that with increasing number of crossing cuts, the number of composed cuts grows even further.

We now calculate the expected similarity for non-crossing cuts and use it as an approximation for the unexpected similarity when cuts cross.

Let $k := |A_\rho(G_1)|$, $l := |A_\rho(G_2)|$, and \mathcal{F}_x denote all approximation sets that contain x cuts. Then by Lemma 2 of [1] we have

$$\begin{aligned} Es(k, l) &= \frac{1}{|\mathcal{F}_k||\mathcal{F}_l|} \sum_{\substack{F_1 \in \mathcal{F}_k \\ F_2 \in \mathcal{F}_l}} |F_1 \cap F_2| = \frac{1}{|\mathcal{F}_k||\mathcal{F}_l|} \sum_{s \in S} |\{F \in \mathcal{F}_k | s \in F\}| \cdot |\{F \in \mathcal{F}_l | s \in F\}| \\ &= \frac{1}{\left\{ \begin{smallmatrix} n \\ k+1 \end{smallmatrix} \right\} \left\{ \begin{smallmatrix} n \\ l+1 \end{smallmatrix} \right\}} \cdot \frac{\sum_{i=1}^{n-1} \left(\binom{n}{i} \sum_{j=0}^{k-1} \left(\left\{ \begin{smallmatrix} i \\ j+1 \end{smallmatrix} \right\} \left\{ \begin{smallmatrix} n-i \\ k-j \end{smallmatrix} \right\} \right) \cdot \sum_{j=0}^{l-1} \left(\left\{ \begin{smallmatrix} i \\ l+1 \end{smallmatrix} \right\} \left\{ \begin{smallmatrix} n-i \\ l-j \end{smallmatrix} \right\} \right) \right)}{(2^{k+1} - 2)(2^{l+1} - 2)}. \end{aligned}$$

The factors $2^{k+1} - 2$ and $2^{l+1} - 2$ prevent from double-counting by choosing the same cuts in a different order.

Deriving a closed formula for Es seems difficult, due to the Stirling numbers of the second kind. We can, however, evaluate the expression algorithmically for every necessary k and l . In order to avoid straight-forward $O(n^7)$ computation, we pre-compute all binomial coefficients from $\binom{n}{1}$ to $\binom{n}{n-1}$ in linear time using the identity $\binom{n}{i+1} = \binom{n}{i} \cdot \frac{(n-i)}{(i+1)}$. Similarly, using the combinatorial identity $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = k \left\{ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\} + \left\{ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\}$, we can pre-compute all Stirling numbers from $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right\}$ to $\left\{ \begin{smallmatrix} n \\ n \end{smallmatrix} \right\}$ in time $O(n^2)$. Using the previously pre-computed values, we can compute all inner summands for different values of l and k in $O(n^3)$ time and space. An evaluation of the formula for two particular values of k and l thus needs only $O(n)$ time and the evaluation for all possible pairs of k and l thus fits into the $O(n^3)$ time necessary for the pre-computations.

4 Experimental results

In order to evaluate the performance of the approximation set optimization for this problem, we tested it on various sets of input instances and compared the performance to two other algorithms. The first being an algorithm where we average edge weights edge by edge and compute minimum cut on the resulting graph and the second being an algorithm where we increase ρ until the intersection of ρ -approximation sets is non-empty for the first time and we choose the cut from this intersection. We look at this second algorithm because it intuitively seems to be a very good approach.

4.1 Tests

Every test is as follows. Three complete, undirected, weighted graph instances are taken as input, where the first two are used to predict a good solution for a future one. Then, this solution is tested against the third instance. Figure 2 illustrates all tests done.

Input: Three complete, undirected, weighted graphs, G_1 , G_2 , and G_3 .

Output: Four different results:

- AVERAGE: Add the edge weights of G_1 and G_2 pairwise. Compute a minimum cut for the new formed graph. Apply the solution on G_3 .
- FIRSTINTERSECTION: Find the smallest ρ that results in a non-empty intersection of the ρ -approximation sets for G_1 and G_2 . Pick a random cut from the intersection and apply it on G_3 .
- BESTSIMILARITY: Find ρ^* which maximizes the unexpected similarity of G_1 and G_2 . Pick a random cut from the intersection and apply it on G_3 .
- OPTIMUM: Compute a minimum cut of G_3 .

Fig. 2: Specification of the Experiment

4.2 Data

We run experiments on three different kind of graphs to evaluate the quality of the found solution: On graphs constructed with real world data which we expect to be similar, on totally random graphs which we do not expect to be similar at all, and on artificially generated similar random graphs, which all have some small cuts in common. The tests on real world data are based on the historical daily prices between 1999 and 2010 of thirteen different stock indices¹ [18]. The vertices of our graph correspond to individual stock indices and the edges between them correspond to their similarity with respect to the problem of finding a contiguous sub-array of maximum sum², as calculated by the approximation set optimization method [1]. Every graph corresponds to one year. For the random graphs, we assign a random weight to every edge. For the artificially made similar graphs we randomly define some cuts to be small and allocate small weights to their edges. To all the other edges we randomly assigned a weight from a larger range.

¹ BEL-20, Dow Jones, Hang Seng, Nikkei, AEX, CAC-40, Dax, Eurotop100, FTSE100, JSX, Nasdaq, AS30, RTSIndex, SMI

² In other words, finding out when to buy and when to sell in order to maximize profit, if we are only allowed to do each operation once.

4.3 Results

Real World Data. To overcome the need of sampling for very high values of ρ , we took logarithms of the edge weights. In most of the tests ρ^* was thus smaller than 3.0. The results are listed in Figure 3. In addition to results on all tests, we extracted pairs of instances with higher than median unexpected similarity and tried to use only those to predict results. As perhaps the only unexpected result, this did not seem to improve the specificity. It seems that the differences between various years vary too much (which corresponds to our ability to predict market behavior, which is, in general, poor).

	sum of all tests (858)	% of opt	sum of all tests with $U \geq \tilde{U}$ (462)	% of opt
Average	65062.20	188.70%	34826.08	187.28%
First Intersection	63682.42	184.70%	34702.42	186.62%
Best Rho	63116.60	183.06%	34702.42	186.62%
Optimum	34478.63	100.00%	18595.24	100.00%

Fig. 3: Stock Market Data (Logarithmised)

Random Graphs. These experiments are mainly done for control purposes. If the data is truly random, we do not expect any algorithm to hold a significant edge, and indeed, the results reflect this. Note that while no algorithm works well here, we are able to realize that this will be so due to low unexpected similarity between instances. Figure 4 and Figure 5 list the results.

	sum of all tests	% of opt	sum of all tests with $U \geq \tilde{U}$ (260)	% of opt
Average	460763	132.74%	232340	132.91%
First Intersection	459802	132.47%	232688	133.11%
Best Rho	458033	131.96%	232546	133.03%
Optimum	347112	100.00%	174810	100.00%

Fig. 4: Random Graphs of 15 Vertices, Edge Weight Range [0-255]

Similar Random Graphs. With these experiments we wanted to verify our expectation that results improve with increasing similarity of graphs, e.g. the larger the expected value of a random cut gets compared to the expected value of a small cut, the better are our results, see Figures 6, 7, and 8. For fixed small cut cost we get even better results, see Figures 9, 10, and 11.

	sum of all tests (512)	% of opt	sum of all tests with $U \geq \tilde{U}$ (261)	% of opt
Average	1616070	122.16%	820594	121.99%
First Intersection	1612010	121.86%	821715	122.15%
Best Rho	1602646	121.15%	820574	121.99%
Optimum	1322892	100.00%	672683	100.00%

Fig. 5: Random Graphs of 50 Vertices, Edge Weight Range [0-255]

	sum of all tests (512)	% of opt	sum of all tests with $U \geq \tilde{U}$ (259)	% of opt
Average	142105	114.43%	65818	108.95%
First Intersection	139536	112.36%	64596	106.93%
Best Rho	139331	112.20%	64596	106.93%
Optimum	124182	100.00%	60410	100.00%

Fig. 6: Similar Graphs with Small Range [0,31] and Big Range [0,255]

	sum of all tests (512)	% of opt	sum of all tests with $U \geq \tilde{U}$ (286)	% of opt
Average	132521	116.03%	72361	113.60%
First Intersection	131285	114.95%	70983	111.44%
Best Rho	128573	112.57%	70983	111.44%
Optimum	114213	100.00%	63697	100.00%

Fig. 7: Similar Graphs with Small Range [0,31] and Big Range [0,127]

	sum of all tests (512)	% of opt	sum of all tests with $U \geq \tilde{U}$ (258)	% of opt
Average	126123	119.87%	61841	116.91%
First Intersection	126831	120.54%	61923	117.06%
Best Rho	123126	117.02%	61581	116.42%
Optimum	105220	100.00%	52897	100.00%

Fig. 8: Similar Graphs with Small Range [0,31] and Big Range [0,63]

	sum of all tests (512)	% of opt	sum of all tests with $U \geq \tilde{U}$ (401)	% of opt
Average	79607	110.85%	57501	104.94%
First Intersection	78311	109.04%	57603	105.13%
Best Rho	77953	108.54%	57573	105.08%
Optimum	71818	100.00%	54792	100.00%

Fig. 9: Similar Graphs with Small Cut Weight 240 and Random Weight Range [0,255]

	sum of all tests (512)	% of opt	sum of all tests with $U \geq \tilde{U}$ (402)	% of opt
Average	155440	111.97%	117571	106.03%
First Intersection	155676	112.14%	117530	106.00%
Best Rho	153880	110.84%	117530	106.00%
Optimum	138828	100.00%	110881	100.00%

Fig. 10: Similar Graphs with Small Cut Weight 500 and Random Weight Range [0,255]

	sum of all tests (512)	% of opt	sum of all tests with $U \geq \tilde{U}$ (374)	% of opt
Average	292254	107.90%	217775	104.36%
First Intersection	291094	107.47%	217787	104.36%
Best Rho	288942	106.68%	217685	104.32%
Optimum	270853	100.00%	208679	100.00%

Fig. 11: Similar Graphs with Small Cut Weight 1000 and Random Weight Range [0,255]

5 Conclusion

We showed how to apply approximation set optimization to the problem of looking for a minimum cut in a graph by adapting a known minimum cut algorithm and estimating the expected intersection of two sets of small cuts.

In general, the experimental results reaffirm our expectation that the algorithm is better at generalizing than other simple heuristic algorithms. In addition to this, the unexpected similarity gives us additional information about the usefulness of our result. In some applications this can be a significant benefit. Having information about the quality of the calculated solution may be very important, in particular when the calculated solution is far from optimal.

By the choice of the optimal parameter ρ , our approach selects a set of minimum cuts which are expected to have low weight in the following graph instances. This can be of significant help as it divides the solution space into sets of relevant and irrelevant cuts, for instance, in a network robustness scenario, it separates the cuts that are likely to be critical from those that are not.

References

1. Joachim M. Buhmann, Matúš Mihalák, Rastislav Šrámek, and Peter Widmayer. Robust optimization in the presence of uncertainty. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science, ITCS '13*, pages 505–514, New York, NY, USA, 2013. ACM.
2. Rodrigo A. Botafogo. Cluster analysis for hypertext systems. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 116–125. ACM, 1993.

3. Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124–1137, 2004.
4. Aparna Ramanathan and Charles J Colbourn. Counting almost minimum cutsets with reliability applications. *Mathematical Programming*, 39(3):253–261, 1987.
5. Johannes J. Schneider and Scott Kirkpatrick. *Stochastic Optimization*. Springer, 2007.
6. Peter Kall and János Mayer. *Stochastic Linear Programming: Models, Theory, and Computation*. Springer Verlag, 2005.
7. Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.
8. Yonatan Bilu and Nathan Linial. Are stable instances easy? In *Proceedings of the First Symposium on Innovations in Computer Science (ICS)*, pages 332–341, 2010.
9. Davide Bilò, Hans-Joachim Böckenhauer, Juraž Hromkovič, Richard Kráľovič, Tobias Mömke, Peter Widmayer, and Anna Zych. Reoptimization of steiner trees. In *SWAT*, pages 258–269, 2008.
10. Matúš Mihalák, Marcel Schöngens, Rastislav Šrámek, and Peter Widmayer. On the complexity of the metric TSP under stability considerations. In *SOFSEM*, pages 382–393, 2011.
11. Matúš Mihalák and Rastislav Šrámek. Counting approximately-shortest paths in directed acyclic graphs. <http://arxiv.org/abs/1304.6707>, 2013.
12. E. A. Dinits, A. V. Karzanov, and V. Lomonosov. *On the Structure of a Family of Minimal Weighted Cuts in a Graph*. 1976.
13. David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-out algorithm. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 21–30, 1993.
14. J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.
15. Hiroshi Nagamochi, Kazuhiro Nishimura, and Toshihide Ibaraki. Computing all small cuts in an undirected network. *SIAM Journal on Discrete Mathematics*, 10(3):469–481, 1997.
16. David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, July 1996.
17. Barbara Geissmann. Approximation set optimization for minimum cut. <http://www.100acrewood.org/~rasto/publications/ThesisBarbaraGeissmann.pdf>, August 2012.
18. Market rates online. <http://www.marketratesonline.com>, August 2012.